Exercício 0: Legibilidade

Implemente um programa que calcule o nível (representado a partir de uma série, como na escola) aproximado necessário para compreender algum texto, conforme a seguir.

$ ./readability  
Texto: Parabéns! Hoje é seu dia. Você está indo para ótimos lugares! Você está com tudo!  
Grade 3

Níveis de leitura

De acordo com a Scholastic , “Charlotte's Web” de EB White está entre o nível de leitura da segunda e quarta séries, e “The Giver” de Lois Lowry está entre um nível de leitura da oitava série e um nível de leitura da décima segunda série. No entanto, o que significa um livro estar no “nível de leitura da quarta série”?

Bem, em muitos casos, um especialista humano pode ler um livro e tomar uma decisão sobre a série para a qual acha que o livro é mais apropriado. Mas você também pode imaginar um algoritmo tentando descobrir qual é o nível de leitura de um texto.

Então, que tipo de características são características de níveis de leitura mais altos? Bem, palavras mais longas provavelmente se correlacionam com níveis de leitura mais altos. Da mesma forma, frases mais longas provavelmente se correlacionam com níveis mais altos de leitura também. Vários “testes de legibilidade” foram desenvolvidos ao longo dos anos, para fornecer um processo estereotipado para calcular o nível de leitura de um texto.

Um desses testes de legibilidade é o índice Coleman-Liau. O índice Coleman-Liau de um texto é projetado para mostrar qual nível escolar (EUA) é necessário para entender o texto. A fórmula é:

índice = 0,0588 \* L - 0,296 \* S - 15,8

Aqui,**L** é o número médio de letras por 100 palavras no texto e S é o número médio de sentenças por 100 palavras no texto.

Vamos escrever um programa chamado **readability** que pega um texto e determina seu nível de leitura. Por exemplo, se o usuário digitar uma linha do Dr. Seuss:

$ ./readability  
Texto: Congratulations! Today is your day. You're off to Great Places! You're off and away!  
Grade 3

O texto que o usuário inseriu tem 65 letras, 4 sentenças e 14 palavras. 65 letras por 14 palavras é uma média de cerca de 464,29 letras por 100 palavras. E 4 sentenças por 14 palavras é uma média de cerca de 28,57 sentenças por 100 palavras. Conectados à fórmula Coleman-Liau e arredondados para o número inteiro mais próximo, obtemos uma resposta de 3: portanto, esta passagem está em um nível de leitura da terceira série.

Vamos tentar outro:

$ ./readability  
Text: Harry Potter was a highly unusual boy in many ways. For one thing, he hated the summer holidays more than any other time of year. For another, he really wanted to do his homework, but was forced to do it in secret, in the dead of the night. And he also happened to be a wizard.  
Grade 5

Este texto contém 214 letras, 4 frases e 56 palavras. Isso resulta em cerca de 382,14 letras por 100 palavras e 7,14 frases por 100 palavras. Conectados à fórmula Coleman-Liau, chegamos ao nível de leitura da quinta série.

À medida que o número médio de letras e palavras por frase aumenta, o índice de Coleman-Liau dá ao texto um nível de leitura mais alto. Se você pegasse este parágrafo, por exemplo, que tem palavras e sentenças mais longas do que qualquer um dos dois exemplos anteriores, a fórmula daria ao texto um nível de leitura de décimo primeiro ano.

$ ./readability  
Text: As the average number of letters and words per sentence increases, the Coleman-Liau index gives the text a higher reading level. If you were to take this paragraph, for instance, which has longer words and sentences than either of the prior two examples, the formula would give the text an eleventh grade reading level.  
Grade 11

Tente!

Este vídeo irá te ajudar a entender o problema ;)

**Atenção:** *para adicionar legendas ao vídeo clique no botão CC localizado no Player e selecione a opção "Português (Brasil)".*

<https://www.youtube.com/watch?v=vTsVt_kDS2M&ab_channel=Funda%C3%A7%C3%A3oEstudar>

Especificação

Projete e implemente um programa, **readability**, que calcule o índice Coleman-Liau do texto.

* Implemente seu programa em um arquivo denominado **readability.c** em um diretório denominado **readability**.
* Seu programa deve solicitar ao usuário uma **string**de texto (usando **get\_string** ).
* Seu programa deve contar o número de letras, palavras e frases do texto. Você pode assumir que uma letra é qualquer caractere minúsculo de **a** a **z** ou qualquer caractere maiúsculo de **A** a **Z** , qualquer sequência de caracteres separados por espaços deve contar como uma palavra e que qualquer ocorrência de um ponto final, ponto de exclamação ou ponto de interrogação indica o final de uma frase.
* Seu programa deve imprimir como saída "**Grade X**", onde **X** é o nível de grau calculado pela fórmula de Coleman-Liau, arredondado para o número inteiro mais próximo.
* Se o número do índice resultante for 16 ou superior (equivalente ou superior ao nível de leitura de graduação sênior), seu programa deve produzir "**Grade 16+**" em vez de fornecer o número do índice exato. Se o número do índice for menor que 1, seu programa deve imprimir "**Before Grade 1**" .

**Como testar seu código no IDE do CS50?**

Aqui alguns casos de teste. Seu código funciona conforme prescrito quando você insere...:

* One fish. Two fish. Red fish. Blue fish. (Before Grade 1)
* Would you like them here or there? I would not like them here or there. I would not like them anywhere. (Grade 2)
* Congratulations! Today is your day. You're off to Great Places! You're off and away! (Grade 3)
* Harry Potter was a highly unusual boy in many ways. For one thing, he hated the summer holidays more than any other time of year. For another, he really wanted to do his homework, but was forced to do it in secret, in the dead of the night. And he also happened to be a wizard. (Grade 5)
* In my younger and more vulnerable years my father gave me some advice that I've been turning over in my mind ever since. (Grade 7)
* Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, "and what is the use of a book," thought Alice "without pictures or conversation?" (Grade 8)
* When he was nearly thirteen, my brother Jem got his arm badly broken at the elbow. When it healed, and Jem's fears of never being able to play football were assuaged, he was seldom self-conscious about his injury. His left arm was somewhat shorter than his right; when he stood or walked, the back of his hand was at right angles to his body, his thumb parallel to his thigh. (Grade 8)
* There are more things in Heaven and Earth, Horatio, than are dreamt of in your philosophy. (Grade 9)
* It was a bright cold day in April, and the clocks were striking thirteen. Winston Smith, his chin nuzzled into his breast in an effort to escape the vile wind, slipped quickly through the glass doors of Victory Mansions, though not quickly enough to prevent a swirl of gritty dust from entering along with him. (Grade 10)
* A large class of computational problems involve the determination of properties of graphs, digraphs, integers, arrays of integers, finite families of finite sets, boolean formulas and elements of other countable domains. (Grade 16+)

Execute o seguinte para avaliar se seu código está correto usando **check50**. Mas certifique-se de compilar e testar você mesmo!

check50 cs50/problems/2021/x/readability

Execute o seguinte para avaliar o style do seu código usando **style50**.

style50 readability.c

Exercício 1: Caesar

Implemente um programa que criptografa mensagens usando a cifra de César, conforme a seguir.

$ ./caesar 13  
plaintext: HELLO  
ciphertext: URYYB

**Lembre-se que plaintext significa *texto simples* e ciphertext é o *texto cifrado*!**

**Background**

Supostamente, César (sim, aquele César) costumava “criptografar” (ou seja, ocultar de forma reversível) mensagens confidenciais deslocando cada letra nelas em algum número de lugares. Por exemplo, ele pode escrever A como B, B como C, C como D, ..., e, agrupando em ordem alfabética, Z como A. E então, para dizer HELLO para alguém, César pode escrever IFMMP. Ao receber essas mensagens de César, os destinatários teriam que “decifrá-las” deslocando as letras na direção oposta no mesmo número de lugares.

O segredo desse “criptosistema” dependia apenas de César e dos destinatários saberem de um segredo, o número de lugares pelos quais César havia mudado suas cartas (por exemplo, 1). Não é particularmente seguro para os padrões modernos, mas, ei, se você é talvez o primeiro no mundo a fazer isso, bastante seguro!

O texto não criptografado é geralmente chamado de texto simples . O texto criptografado é geralmente chamado de texto cifrado . E o segredo usado é chamado de chave .

Para ser claro, então, aqui está como criptografar **HELLO**com uma chave de 1 resulta em **IFMMP**:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| plaintext (texto simples) | H | E | L | L | O |
| + key(chave) | 1 | 1 | 1 | 1 | 1 |
| = ciphertext (texto cifrado) | I | F | M | M | P |

Mais formalmente, o algoritmo de César (isto é, cifra) criptografa mensagens “girando” cada letra em k posições. Mais formalmente, se p é algum texto simples (ou seja, uma mensagem não criptografada), pi é o i-ésimo caractere em p , e k é uma chave secreta (ou seja, um inteiro não negativo), então cada letra, c i , em o texto cifrado, c , é calculado como

c i = (p i + k)% 26

em que**%26** aqui significa "resto ao dividir por 26." Essa fórmula talvez faça a cifra parecer mais complicada do que é, mas na verdade é apenas uma maneira concisa de expressar o algoritmo com precisão. De fato, para fins de discussão, pense em A (ou a) como 0, B (ou b) como 1, ..., H (ou h) como 7, I (ou i) como 8, ... e Z (ou z) como 25. Suponha que César apenas queira dizer Oi para alguém confidencialmente usando, desta vez, uma chave, k , de 3. E então seu texto simples, p , é Hi, em cujo caso o primeiro caractere de seu texto simples, p 0 , é H (também conhecido como 7), e o segundo caractere de seu texto simples, p 1 , é i (também conhecido como 8). O primeiro caractere de seu texto cifrado, c 0 , é portanto K, e o segundo caractere de seu texto cifrado, c 1 , é assim L. Você pode ver por quê?

Vamos escrever um programa chamado **ceasar**que permite criptografar mensagens usando a cifra de César. No momento em que o usuário executa o programa, ele deve decidir, fornecendo um argumento de linha de comando, qual deve ser a chave na mensagem secreta que fornecerá no tempo de execução. Não devemos necessariamente presumir que a chave do usuário será um número; embora você possa assumir que, se for um número, será um inteiro positivo.

Aqui estão alguns exemplos de como o programa pode funcionar. Por exemplo, se o usuário inserir uma chave de **1** e um texto simples de HELLO :

$ ./caesar 1

plaintext: HELLO

ciphertext: IFMMP

Veja como o programa pode funcionar se o usuário fornecer uma chave **13** e um texto simples de hello, world :

$ ./caesar 13

plaintext: hello, world

ciphertext: uryyb, jbeyq

Observe que nem a vírgula nem o espaço foram "deslocados" pela cifra. Gire apenas caracteres alfabéticos!

Que tal mais um? Veja como o programa pode funcionar se o usuário fornecer uma chave **13** novamente, com um texto simples mais complexo:

$ ./caesar 13

plaintext: be sure to drink your Ovaltine

ciphertext: or fher gb qevax lbhe Binygvar

Observe que a caixa da mensagem original foi preservada. As letras minúsculas permanecem minúsculas e as maiúsculas permanecem maiúsculas.

E se um usuário não cooperar?

$ ./caesar HELLO

Usage: ./caesar key

Ou realmente não coopera?

$ ./caesar

Usage: ./caesar key

Ou mesmo ...

$ ./caesar 1 2 3

Usage: ./caesar key

Este vídeo irá te ajudar a entender o problema ;)

**Atenção:** *para adicionar legendas ao vídeo clique no botão CC localizado no Player e selecione a opção "Português (Brasil)".*

<https://www.youtube.com/watch?v=82pMEmDMnxI&ab_channel=Funda%C3%A7%C3%A3oEstudar>

Especificação

Projete e implemente um programa, o **caesar**, que criptografa mensagens usando a cifra de César.

* Implemente seu programa em um arquivo denominado **caesar.c** em um diretório denominado **caesar**.
* Seu programa deve aceitar um único argumento de linha de comando, um inteiro não negativo. Vamos chamá-lo de k para fins de discussão.
* Se o seu programa for executado sem nenhum argumento de linha de comando ou com mais de um argumento de linha de comando, seu programa deve imprimir uma mensagem de erro de sua escolha (com **printf**) e retornar de **main**um valor de **1** (o que tende a significar um erro ) imediatamente.
* Se algum dos caracteres do argumento da linha de comando não for um dígito decimal, seu programa deve imprimir a mensagem Use: ./caesar key e retornar de **main**o valor **1** .
* Não suponha que k será menor ou igual a 26. Seu programa deve funcionar para todos os valores integrais não negativos de k menores que 2 ^ 31 - 26. Em outras palavras, você não precisa se preocupar se seu programa eventualmente quebra se o usuário escolher um valor para k que é muito grande ou quase grande para caber em um **int**. (Lembre-se de que um **int**pode estourar.) Mas, mesmo se k for maior que 26, os caracteres alfabéticos na entrada do programa devem permanecer caracteres alfabéticos na saída do programa. Por exemplo, se k é 27, A não deve se tornar**[** embora**[** esteja a 27 posições de A em ASCII, por http://www.asciichart.com/[asciichart.com]; A deve tornar-se B , já que B esta a 27 posições de A , desde que você revolva em torno de Z a A.
* Seu programa deve produzir **plaintext:** (sem uma nova linha) [texto simples] e então solicitar ao usuário uma **string**de texto simples (usando **get\_string** ).
* Seu programa deve produzir **ciphertext:** (sem uma nova linha) [texto cifrado] seguido pelo texto cifrado correspondente do texto simples, com cada caractere alfabético no texto simples “girado” por k posições; os caracteres não alfabéticos devem ser reproduzidos inalterados.
* Seu programa deve preservar as letras maiúsculas e minúsculas: as letras maiúsculas, embora giradas, devem permanecer letras maiúsculas; as letras minúsculas, embora giradas, devem permanecer em minúsculas.
* Após a saída do texto cifrado, você deve imprimir uma nova linha. Seu programa deve então sair retornando **0** de **main**.

Como começar? Vamos abordar esse problema um passo de cada vez.

Pseudocódigo

Spoiler ;)

**Existe mais de uma forma para resolver esse exercício, esse spoiler aqui é apenas uma delas!**  
1- Verifique se o programa foi executado com um argumento de linha de comando;  
2- Repita o argumento fornecido para garantir que todos os caracteres sejam dígitos;  
3- Converta o argumento da linha de comando de uma **string**para **int**  
4- Solicite plaintext: (texto simples) ao usuário  
5- Repita/Itere para cada caractere do plaintext: (texto simples)  
    1- Se é uma letra maiuscula, rotacione-a, preservando capitalização, e então imprima o caractere rotacionado.  
    2- Se é uma letra minúscula, rotacione-a, preservando capitalização, e então imprima o caractere rotacionado.  
    3-  Se não for nenhum, imprima o caractere como está  
6- Imprimir uma nova linha  
  
***Não há problema em editar o seu depois de ver este pseudocódigo aqui, mas não simplesmente copie/cole o nosso no seu. Ok?***

**Como testar seu código no IDE do CS50?**

Execute o seguinte para avaliar se seu código está correto usando **check50**. Mas certifique-se de compilar e testar você mesmo!

check50 cs50/problems/2021/x/caesar

Execute o seguinte para avaliar o style do seu código usando **style50**.

style50 caesar.c

Exercício 2: Substituição (desafio)

Implemente um programa que implemente uma cifra de substituição, conforme a seguir.

$ ./substitution JTREKYAVOGDXPSNCUIZLFBMWHQ  
texto simples: HELLO  
texto cifrado: VKXXN

Background

Em uma cifra de substituição, “criptografamos” (ou seja, ocultamos de forma reversível) uma mensagem substituindo cada letra por outra. Para isso, usamos uma chave : neste caso, um mapeamento de cada uma das letras do alfabeto à letra correspondente quando criptografada. Para “decifrar” a mensagem, o receptor da mensagem precisaria saber a chave, para que pudesse reverter o processo: traduzir o texto criptografado (geralmente chamado de texto cifrado ) de volta na mensagem original (geralmente chamado de texto simples ).

Uma chave, por exemplo, pode ser a sequencia NQXPOMAFTRHLZGECYJIUWSKDVB. Esta chave de 26 caracteres significa que **A**(a primeira letra do alfabeto) deve ser convertido em **N** (o primeiro caractere da chave), **B** (a segunda letra do alfabeto) deve ser convertido em **Q** (o segundo caractere do chave), e assim por diante.

Uma mensagem como **HELLO** , então, seria criptografada como **FOLLE** , substituindo cada uma das letras de acordo com o mapeamento determinado pela chave.

Vamos escrever um programa chamado **substitution** que permite criptografar mensagens usando uma cifra de substituição. No momento em que o usuário executa o programa, ele deve decidir, fornecendo um argumento de linha de comando, qual deve ser a chave na mensagem secreta que fornecerá durante a execução.

Aqui estão alguns exemplos de como o programa pode funcionar. Por exemplo, se o usuário inserir uma chave de **YTNSHKVEFXRBAUQZCLWDMIPGJO** e um texto simples de **HELLO** :

$ ./substitution YTNSHKVEFXRBAUQZCLWDMIPGJO  
texto simples: HELLO  
texto cifrado: EHBBQ

Veja como o programa pode funcionar se o usuário fornecer uma chave de **VCHPRZGJNTLSKFBDQWAXEUYMOI** e um texto simples de**hello, world** :

$ ./substitution VCHPRZGJNTLSKFBDQWAXEUYMOI  
texto simples: hello, world  
texto cifrado: jrssb, ybwsp

Observe que nem a vírgula nem o espaço foram substituídos pela cifra. Substitua apenas caracteres alfabéticos! Observe também que o case da mensagem original foi preservado. As letras minúsculas permanecem minúsculas e as maiúsculas permanecem maiúsculas.

Não importa se os caracteres da própria chave são maiúsculas ou minúsculas. Uma chave de **VCHPRZGJNTLSKFBDQWAXEUYMOI** é funcionalmente idêntica a uma chave de **vchprzgjntlskfbdqwaxeuymoi** (como é, nesse caso, **VcHpRzGjNtLsKfBdQwAxEuYmOi** ).

E se um usuário não fornecer uma chave válida?

$ ./substitution ABC  
A chave deve conter 26 caracteres.

Ou realmente não coopera?

$ ./substitution  
Uso: ./ chave de substituição

Ou mesmo ...

$ ./substituição 1 2 3  
Uso: ./ chave de substituição

Este vídeo irá te ajudar a entender o problema ;)

**Atenção:** *para adicionar legendas ao vídeo clique no botão CC localizado no Player e selecione a opção "Português (Brasil)".*

<https://www.youtube.com/watch?v=yOGX9KxvYnA>

Especificação

Projete e implemente um programa, **substitution**, que criptografa mensagens usando uma cifra de substituição.

* Implemente seu programa em um arquivo denominado **substitution.c** em um diretório denominado **substitution** .
* Seu programa deve aceitar um único argumento de linha de comando, a chave a ser usada para a substituição. A chave em si não deve fazer distinção entre maiúsculas e minúsculas, portanto, se algum caractere na chave estiver em maiúscula ou minúscula, isso não deve afetar o comportamento do seu programa.
* Se o seu programa for executado sem nenhum argumento de linha de comando ou com mais de um argumento de linha de comando, seu programa deve imprimir uma mensagem de erro de sua escolha (com **printf**) e retornar de **main**um valor de **1** (o que tende a significar um erro ) imediatamente.
* Se a chave for inválida (por não conter 26 caracteres, conter qualquer caractere que não seja um caractere alfabético, ou não conter cada letra exatamente uma vez), seu programa deverá imprimir uma mensagem de erro de sua escolha (com **printf**) e retornar do **main**um valor de **1** imediatamente.
* Seu programa deve produzir na saída **plaintext:** - que significa texto simples: (sem uma nova linha) e então solicitar ao usuário uma string de texto simples (usando get\_string ).
* Seu programa ter na saída o seguinte texto **ciphertext:** - que significa texto cifrado - (sem uma nova linha) seguido pelo texto cifrado correspondente do texto simples, com cada caractere alfabético no texto simples substituído pelo caractere correspondente no texto cifrado; os caracteres não alfabéticos devem ser reproduzidos inalterados.
* Seu programa deve preservar maiúsculas e minúsculas: as letras maiúsculas devem permanecer letras maiúsculas; as letras minúsculas devem permanecer em minúsculas.
* Após a saída do texto cifrado, você deve imprimir uma nova linha. Seu programa deve então sair retornando **0** da **main**.

**Como testar seu código no IDE do CS50?**

Execute o seguinte para avaliar se seu código está correto usando **check50**. Mas certifique-se de compilar e testar você mesmo!

check50 cs50/problems/2021/x/substitution

Execute o seguinte para avaliar o style do seu código usando **style50**.

style50 substitution.c

## Lab 2: Scrabble

Determine qual das duas palavras do Scrabble vale mais.

$ ./scrabble  
Player 1: COMPUTER  
Player 2: science  
Player 1 wins!

## Background

No jogo [**Scrabble**](https://scrabble.hasbro.com/en-us/rules), os jogadores criam palavras para marcar pontos, e o número de pontos é a soma dos valores dos pontos de cada letra da palavra.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **D** | **E** | **F** | **G** | **H** | **I** | **J** | **K** | **L** | **M** | **N** | **O** | **P** | **Q** | **R** | **S** | **T** | **U** | **V** | **W** | **X** | **Y** | **Z** |
| 1 | 3 | 3 | 2 | 1 | 4 | 2 | 4 | 1 | 8 | 5 | 1 | 3 | 1 | 1 | 3 | 10 | 1 | 1 | 1 | 1 | 4 | 4 | 8 | 4 | 10 |

Por exemplo, se quiséssemos pontuar a palavra **Code**, observaríamos que, nas regras gerais do Scrabble, o **C** vale **3** pontos, o **o** vale **1**ponto, o **d** vale**2** pontos e**e** vale **1** ponto. Somando isso, obtemos que **Code** vale **3 + 1 + 2 + 1 = 7** pontos.

<https://www.youtube.com/watch?v=6Wn1eb0eE_4&ab_channel=Funda%C3%A7%C3%A3oEstudar>

## Começando

Copie o “código de distribuição” (ou seja, código inicial) a seguir em um novo arquivo em seu IDE chamado **scrabble.c**.

#include <ctype.h>

#include <cs50.h>

#include <stdio.h>

#include <string.h>

// Points assigned to each letter of the alphabet

int POINTS[] = {1, 3, 3, 2, 1, 4, 2, 4, 1, 8, 5, 1, 3, 1, 1, 3, 10, 1, 1, 1, 1, 4, 4, 8, 4, 10};

int compute\_score(string word);

int main(void)

{

// Get input words from both players

string word1 = get\_string("Player 1: ");

string word2 = get\_string("Player 2: ");

// Score both words

int score1 = compute\_score(word1);

int score2 = compute\_score(word2);

// TODO: Print the winner

}

int compute\_score(string word)

{

// TODO: Compute and return score for string

}

* Você também pode fazer download do código de distribuição executando o comando **wget https://cdn.cs50.net/2020/fall/labs/2/scrabble.c** no CS50 IDE.

## Detalhes de Implementação

Conclua a implementação de scrabble.c, de modo que determine o vencedor de um jogo curto do tipo scrabble, em que dois jogadores digitam suas palavras, e o jogador com maior pontuação vence.

Observe que armazenamos os valores dos pontos de cada letra do alfabeto em uma matriz de inteiros chamada **POINTS**.

* Por exemplo, **A** ou**a** vale **1** ponto (representado por **POINTS[0]**), **B** ou **b** vale **3** pontos (representado por **POINTS[1]**), etc.

Observe que criamos um protótipo para uma função auxiliar chamada **compute\_score()** que recebe uma string como input e retorna um **int.** Sempre que quisermos atribuir valores de pontos a uma palavra específica, podemos chamar essa função. Observe que este protótipo é necessário para o C saber que **compute\_score()** existe posteriormente no programa.

Em main() , o programa pede aos dois jogadores suas palavras usando a função **get\_string()**. Esses valores são armazenados dentro de variáveis ​​chamadas **word1** e **word2**.

Em c**ompute\_score()**, seu programa deve calcular, usando o array **POINTS**, e retornar a pontuação para o argumento string. Os caracteres que não são letras devem receber zero pontos, e as letras maiúsculas e minúsculas devem receber os mesmos valores de pontos.

* Por exemplo, **!** vale **0** pontos, enquanto **A** e a valem **1** ponto.
* Embora as regras do Scrabble normalmente exijam que uma palavra esteja no dicionário, não há necessidade de verificar isso neste problema!

Em **main()**, seu programa deve imprimir, dependendo da pontuação dos jogadores, **Player 1 wins!**, **Player 2 wins!**, ou **Tie!**.

## Dicas

Você pode achar as funções **isupper()** e **islower()** úteis para você. Essas funções usam um caractere como argumento e retornam um valor diferente de zero se o caractere for maiúsculo (para **isupper** ) ou minúsculo (para **islower**).

Para encontrar o valor no n -ésimo índice de um array chamado **arr**, podemos escrever **arr[n]** . Podemos aplicar isso a strings também, já que strings são arrays de caracteres.

Lembre-se de que os computadores representam caracteres usando [ASCII](https://www.asciitable.com/), um padrão que representa cada caractere como um número.

## Como testar seu código

Seu código deve resolver os seguintes casos de teste:

* $ ./scrabble
* Player 1: Question?
* Player 2: Question!
* Tie!
* $ ./scrabble
* Player 1: Oh,
* Player 2: hai!
* Player 2 wins!
* $ ./scrabble
* Player 1: COMPUTER
* Player 2: science
* Player 1 wins!
* $ ./scrabble
* Player 1: Scrabble
* Player 2: wiNNeR
* Player 1 wins!

Execute o seguinte comando para avaliar a exatidão do seu código usando **check50**. Mas certifique-se de compilar e testar você mesmo!

check50 cs50/labs/2021/x/scrabble

Execute o seguinte comando para avaliar o estilo do seu código usando **style50**.

style50 scrabble.c